ASP.NET 4.0 Output Caching

Een gedetailleerde blik op vernieuwde mogelijkheden

Michiel van Otegem

Output Caching is een onderdeel van ASP.NET dat maar weinig bekendheid geniet. En dat terwijl Output Caching applicaties sneller en schaalbaarder kan maken zonder dat dit veel inspanning vergt. Door de nieuwe caching architectuur van ASP.NET 4.0 is Output Caching nog interessanter geworden. We kijken daarom in dit artikel met een frisse blik naar Output Caching en wat de nieuwe architectuur voor gevolgen heeft.

Wanneer je gebruik maakt van Output Caching wordt het HTML-resultaat van een pagina in de cache opgeslagen. Volgende aanvragen voor dezelfde pagina worden uit de cache geserveerd, waardoor de pagina niet opnieuw moet worden uitgevoerd. Vooral pagina's die veel verwerking vergen en voor alle gebruikers hetzelfde zijn kunnen hier enorm van profiteren. Denk bijvoorbeeld aan de homepage van een nieuwssite die duizenden malen per minuut wordt opgevraagd. Output Caching in ASP.NET werkt heel eenvoudig. Je voegt bovenin de pagina een Output Cache aanwijzing in die aangeeft hoe de cache met moet werken voor de pagina en je bent klaar. Een simpele aanwijzing ziet er als volgt uit:

```
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

Deze aanwijzing geeft aan dat de pagina 60 seconden in de cache mag staan en dat het resultaat niet verschilt afhankelijk van parameters in de querystring of in een formulier. De OutputCache-aanwijzing heeft nog veel meer mogelijkheden dan in het voorbeeld hierboven. Deze zijn grofweg te verdelen in drie categorieën: wat, welke en hoe.

Wat moet in de cache?

ASP.NET is dynamisch en dat betekent dat verschillende soorten aanvragen voor een pagina verschillende resultaten kunnen opleveren. De vraag is dus wat er eigenlijk in de cache moet komen en wanneer een pagina uit de cache moet worden geserveerd. Om dit te controleren kun je gebruik maken van een aantal instellingen die aangeven onder welke voorwaarden er verschillende resultaten van een pagina opgeslagen mogen worden in de cache. Omdat er minimaal moet worden aangegeven of er überhaupt variatie is, heb je al kennis gemaakt met VaryByParam. Deze instelling kan drie verschillende waardes bevatten. Met de waarde None geef je aan dat er geen variatie is ongeacht of er verschillen zijn in de querystring of in waardes die met POST gestuurd zijn.

De volgende URLs zijn dus voor de cache gelijk:

- http://voorbeeld.nl/demo.aspx;
- http://voorbeeld.nl/demo.aspx?a=1;
- http://voorbeeld.nl/demo.aspx?a=1&b=1.

Omgekeerd zorgt de waarde '*' ervoor dat voor elk van de bovenstaande URLs een afzonderlijke versie in de cache wordt gezet. Alleen aanvragen met exact dezelfde URL worden als identiek beschouwd. Tot slot kun je ook de namen opgeven van de parameters die voor verschillende versies zorgen. Als je de waarde 'a' daar op zou geven, zouden de tweede en de derde URL als identiek beschouwd worden, omdat de parameter 'b' genegeerd wordt door de cache.

Verwant aan VaryByParam zijn VaryByHeaders en VaryByContentEncodings. De eerste varieert op basis van de HTTP headers die in een aanvraag worden gestuurd en de tweede varieert op basis van de Accept-Encoding headers. In beide gevallen dien je een puntkomma-gescheiden lijst op te geven van headers of encodings die voor variatie zorgen. Het volgende voorbeeld zou bijvoorbeeld variëren per taal en de browser's User Agent string:

```
<%@ OutputCache Duration="60" VaryByParam="None"
 VaryByHeaders="Accept-Language;User-Agent" %>
```

Jammer genoeg kan het nog steeds nodig zijn om per browser andere output te genereren, maar hoewel het bovenstaande dat zou doen, zou dat voor elke verschillende User-Agent string een aparte versie opslaan in de cache. Ook als het verschil alleen maar de CLR versie is die geïnstalleerd is op de client. Daarom is het beter om VaryByCustom toe te passen voor verschillen tussen browsers. Geef je die de waarde 'browser', dan wordt per browserversie een afzonderlijk resultaat in de cache geplaatst. Het gaat dan om de major version, zodat minieme versieverschillen niet leiden tot aparte resultaten in de cache. Indien je gebruik maakt van VaryByCustom, kun je in global.asax de methode GetVaryByCustom-

String overschrijven die bepaalt wanneer er variatie moet zijn. Deze methode krijgt een parameter mee waarvoor de variatie moet zijn en dient een string te retourneren die aangeeft tot welke variatie het element behoort. Op die manier zou je bijvoorbeeld voor een Nederlandse browser Nederlandse content in de cache laten opslaan en voor alle andere Engels, zoals het voorbeeld in Listing 1. Daar zie je dat wanneer de parameter custom de waarde 'taal' heeft er een verschil wordt gemaakt tussen Nederlands en Engels. Je kunt hier uiteraard ook voor andere waardes een functie toevoegen. Je gebruikt Listing 1 als volgt in een cache-aanwijzing.

```
<%@ OutputCache Duration="60" VaryByParam="None"
 VaryByCustom="taal" %>
```

```
public override string GetVaryByCustomString(HttpContext context,
string custom)
{
  if (String.IsNullOrEmpty(custom) == false &&
    custom.Equals("TAAL", StringComparison.InvariantCultureIgnore-
Case))
  {
if (context.Request.UserLanguages[0].StartsWith("nl")) return "nl";
    return "en";
  }
  return base.GetVaryByCustomString(context, custom);
}
```

LISTING 1: VARYBYCUSTOM OP TAAL.

Welke cache?

Caching hoeft niet per se aan de kant van de server te gebeuren. De browser cache en proxy caches die zich tussen de client en de server bevinden zijn notoire boosdoeners voor ongewenste resultaten. De OutputCache-aanwijzing bevat daarom niet alleen eigenschappen om te bepalen hoe de cache op de server werkt, maar ook eigenschappen waarmee je kunt aangeven of er ook de browser of een proxy het resultaat in een cache mag plaatsen. In feite controleer je hiermee welke HTTP headers verstuurd worden die invloed uitoefenen op browser en proxy caching. Helaas is er geen 100% garantie dat browsers en proxies de aanwijzingen in de headers opvolgen. Dat maakt het echter niet minder belangrijk om te weten wat de standaardinstellingen zijn en hoe je deze in ASP.NET kunt aanpassen.

De OutputCache-aanwijzing bevat twee eigenschappen om te beïnvloeden in welke cache gegevens wel of niet opgeslagen mogen worden. De eenvoudigste is NoStore, die true of false (standaard) als waarde kan hebben. Indien de waarde true is, voegt ASP.NET de HTTP header Cache-Control: no-store in. Deze geeft aan dat proxies en de browser de pagina niet mogen opslaan. In de HTTP specificatie staat zelfs expliciet dat er geen opslag plaats mag vinden op opslagmedia (disk, tape etc.) en dat als het nodig is om het in het geheugen te plaatsen, het geheugen zo snel mogelijk gewist moet worden. Dit is vooral bedoeld om te voorkomen dat gevoelige informatie per ongeluk nog ergens in geheugen of opslag staat. Dit impliceert ook dat de pagina niet in de ASP.NET cache wordt geplaatst.

De eigenschap Location geeft wat meer mogelijkheden dan NoStore. Je kunt hiermee aangeven waar een pagina in de cache opgenomen mag worden. De verschillende locaties zijn de server, de browser en alles wat zich daar tussenin bevindt. In tabel 1 zie je de verschillende opties voor de eigenschap Location. Belangrijk hierin is ook het gebruik van de HTTP header Cache-Control die drie waardes kan hebben die van belang zijn: no-cache, private en public. In het eerste geval mag het resultaat niet in een cache geplaatst worden en forceer je dus effectief dat een browser altijd een nieuwe aanvraag doet bij de server. Dit is van belang voor pagina's die altijd veranderen. Wordt een pagina als private aangemerkt, dan mag alleen de browser de pagina in de cache plaatsen. Indien een pagina als public wordt aangemerkt dan mag deze in de browser, de server en in proxies in de cache gezet worden.

Waarde	Omschrijving
None	Er vindt geen caching plaats en de server stuurt de header Cache-Control: no-cache mee.
Server	De pagina wordt in de server cache geplaatst en de server stuurt de header Cache-Control: no-cache mee om caching op andere plaatsen te voorkomen.
Client	Er vindt geen caching plaats en de server stuurt de header Cache-Control: private mee.
ServerAndClient	Combinatie van Server en Client.
Downstream	Er vindt geen caching plaats op de server, maar proxies en browsers mogen de pagina wel in een cache plaatsen, overeenkomstig met de header Cache-Control: public.
Any	De pagina kan op iedere plaats in de cache geplaatst worden.

TABEL 1: MOGELIJKHEDEN VOOR DE OUTPUTCACHE EIGENSCHAP LOCATION.

Hoe moet de cache werken?

De cache moet voor een bepaalde tijd elementen vanuit de cache serveren. Je kunt dit op twee manieren instellen. Ten eerste kun je de duur opgeven dat een element in de cache mag staan. Je geeft dit op in seconden in de eigenschap Duration. Ten tweeded kun je de cache ook afhankelijk maken van data in een SQL Server tabel. Wanneer de inhoud van de tabel wijzigt worden cache elementen die aan die tabel gekoppeld zijn verwijderd uit de cache. Dit doe je met behulp van de eigenschap SqlDependency die naar een SqlCacheDependency in web.config moet wijzen. Listing 2 laat zien hoe de configuratie eruit ziet. De cache-aanwijzing ziet er dan bijvoorbeeld als volgt uit voor de tabel Products op basis van de MyDb-Cache dependency uit de configuratie:

```
<%@OutputCache Duration="300" VaryByParam="None"
 SqlDependency="MyDb-Cache:Products" />
```

```
<caching>
  <sqlCacheDependency enabled = "true" pollTime = "20000">
    <databases>
    <add name="MyDb-Cache" connectionStringName="MyDb" pollTime =
 "60000"/>
    </databases>
  </sqlCacheDependency>
</caching>
```

LISTING 2: SQLCACHEDEPENDENCY CONFIGURATIE.

Selectief cachen

De OutputCache-aanwijzing werkt ook in een user control. Hierdoor kun je delen van een pagina afzonderlijk vanuit de cache laten serveren (of juist niet). We noemen dit fragment caching. Dat dezelfde control op meerdere pagina's staat wil niet meteen zeggen dat dezelfde control maar één keer in de cache staat. Of het resultaat van een control gedeeld wordt tussen verschillende pagina's

Een manier om gedeeltes van een pagina wel of niet te cachen is door zogenaamde 'donut caching' te gebruiken.

geef je aan met de eigenschap Shared, die standaard op false staat. Wanneer deze de waarde true heeft wordt het resultaat tussen pagina's gedeeld.

Een andere manier om gedeeltes van een pagina wel of niet te cachen is door zogenaamde donut caching te gebruiken. Daarbij wordt een hele pagina in de cache geplaatst, behalve een klein stukje (of meerdere) dat zich op iedere plek van de pagina kan bevinden. Dit werkt met behulp van de Substitution control die je in een pagina kunt plaatsen op plekken waar je dynamische content wilt hebben, terwijl de rest van de pagina uit de cache komt. De Substitution-control heeft één eigenschap en dat is MethodName. Deze eigenschap moet wijzen naar een statische methode die een string retourneert om weer te geven. Deze statische methode wordt in principe iedere keer aangeroepen als de pagina uitgevoerd wordt. Deze mogelijkheid is bijvoorbeeld handig wanneer veel van de pagina hetzelfde is, maar onderdelen gebruikersspecifiek zijn, zoals het tonen van de naam van de ingelogde gebruiker. Een uitgebreide uitleg over donut caching vind je op http://bit.ly/b8OJTn).

Custom Cache Provider

In ASP.NET 4.0 is de manier waarop de cache onder water werkt veranderd. Voorheen was de cache verweven in de ASP.NET pipeline en kon de manier waarop de cache werkt niet aangepast worden. In ASP.NET 4.0 is output caching losgetrokken en configureerbaar gemaakt volgens het Provider Model Design Pattern. Dit is hetzelfde design pattern dat onder andere ook voor Membership en Roles toegepast wordt. Net als voor die functionaliteiten kun je een andere provider gebruiken als de standaardprovider niet doet wat jij verlangt. Standaard maakt de Output Cache gebruik van de AspNetInternalProvider die elementen in een geheugencache plaatst. Dit werkt in feite precies zo als output caching voor ASP.NET 4.0.

Wanneer je een eigen Cache Provider maakt moet je de pagina (of control) zelf ergens opslaan en bijhouden wanneer de cache daarvoor verloopt. In een database kun je hier kolommen van maken, maar anders is het handig om hiervoor een wrapper object te maken zoals te zien in Listing 3. Dit object moet Serializable zijn om het op te kunnen slaan als een string.

```
[Serializable]
internal class CacheItem
{
  public DateTime Expires;
  public object Item;
}
```

LISTING 3: WRAPPER OBJECT VOOR CACHING.

Om een eigen provider te schrijven dien je de OutputCacheProvider class te overerven. Wanneer je in de configuratie extra parameters mee wilt geven, zoals de database connectionstring of het pad op het file systeem waar je de cache elementen op wilt slaan, dan dien je de Initialize methode te overschrijven en daar de betreffende waarde uit een NameValueCollection te halen. Hoe de Initialize methode eruit ziet voor een Cache Provider die op het file systeem bestanden opslaat zie je in Listing 4. Hierin wordt gecontroleerd of de eigenschap path aanwezig is in de configuratie. Daarna wordt ook nog gekeken of het een applicatiepad is dat vertaald moet worden naar een absoluut pad. Een voorbeeld van de configuratie zie je in Listing 5.

```
public override void Initialize(string name, System.Collections.
Specialized.NameValueCollection config)
{
  base.Initialize(name, config);

  CachePath = config["path"];
  if (String.IsNullOrEmpty(CachePath) == true)
  {
    string message = String.Format("'path' property on cache
provider '{0}' may not be empty.", this.Name);
    throw new ConfigurationErrorsException(message);
  }

  if(CachePath.StartsWith("~"))
  {
    HttpContext context = HttpContext.Current;
    if (context != null)
    {
      CachePath = context.Server.MapPath(config["path"]);
    }
  }

  if (CachePath.EndsWith("\\") == false) CachePath += "\\";
}
```

LISTING 4: PROVIDER EIGENSCHAPPEN INLEZEN IN INITIALIZE METHODE.

```
<caching>
  <outputCache defaultProvider="MyProvider">
  <providers>
    <add name="MyProvider" type="DotNetMag.FileCacheProvider,
DotNetMag" path="~/Cache" />
  </providers>
  </outputCache>
</caching>
```

LISTING 5: CONFIGURATIE VAN EEN CUSTOM PROVIDER.

Om de Cache Provider het werk te laten doen, dien je de methodes Add, Get, Remove en Set te overschrijven. Dit is relatief simpel, maar je moet natuurlijk wel de nodige controles uitvoeren met betrekking tot het bestaan en verlopen van elementen in de cache. Listing 6 laat zien hoe de Get methode eruit kan zien voor een cache op het file systeem. De volledige code is beschikbaar als download. Een provider die gebruik maakt van Windows Server AppFabric Caching kun je downloaden van http://aspnet.codeplex.com/releases/view/46576.

```
public override object Get(string key)
{
  String path = GetFullCachePath(key);
  if (File.Exists(path) == false) return null;

  CacheItem item = new CacheItem();
  using (FileStream fs = File.OpenRead(path))
  {
    BinaryFormatter formatter = new BinaryFormatter();
    item = (CacheItem)formatter.Deserialize(fs);
```

```
  }

  if (item == null || item.Expires <= DateTime.Now.ToUniversal-
Time())
  {
    Remove(key);
    return null;
  }
  return item.Item;
}
```

LISTING 6: GET METHODE VAN EEN FILE CACHE.

Wanneer je verschillende cache mogelijkheden geconfigureerd hebt, kun je verschillende user controls in verschillende caches plaatsen. Het voordeel hiervan is dat je de cache kunt kiezen die het beste past bij die control. Een control met weinig variatie en een kleine geheugen footprint kun je dan in het geheugen plaatsen, terwijl een control met veel variatie en grote elementen kun je op disk cachen. In welke cache een user control geplaats moet worden geeft je aan in de OutputCache-aanwijzing, als volgt:

```
<%@ OutputCache Duration="300" VaryByParam="None"
providerName="MyCache" %>
```

Beperkingen van een Custom Provider

Je kunt in principe op ieder moment een andere cache provider inzetten voor output caching. Je moet er echter wel rekening mee houden dat er een verschil is tussen de AspNetInternalProvider die standaard gebruikt wordt en andere providers. Dit is omdat de AspNetInternalProvider in-process werkt. Caching gebeurt dan in het AppDomain van de applicatie en bij custom providers is dit in principe niet het geval. Dat een custom cache provider over de grens van het AppDomain gaat, heeft wel consequenties voor cache validatie. Met cache validatie kun je een methode opgeven die bepaalt of een pagina uit de cache geserveerd moet worden of niet. Met de interne cache van ASP.NET mag dit een instance methode zijn. Met een Custom Cache Provider moet dit een statische methode zijn. De reden hiervoor is dat het AppDomain kan herstarten waardoor referenties naar instance methodes verdwijnen. Statische referenties kunnen echter weer opgebouwd worden als het AppDomain weer in de lucht komt.

Michiel van Otegem, is directeur/chief software architect bij BataviaLabs.